Journal of Discrete Algorithms 16 (2012) 79–89

Contents lists available at SciVerse ScienceDirect

Journal of Discrete Algorithms

www.elsevier.com/locate/jda

A golden ratio parameterized algorithm for Cluster Editing

Sebastian Böcker *

Lehrstuhl für Bioinformatik, Friedrich-Schiller-Universität Jena, Ernst-Abbe-Platz 2, 07743 Jena, Germany

ARTICLE INFO

Article history:
Available online 3 April 2012

Keywords:
Cluster editing
Parameterized algorithm
FPT
Search tree algorithm
Computational complexity

ABSTRACT

The Cluster Editing problem asks to transform a graph by at most k edge modifications into a disjoint union of cliques. The problem is NP-complete, but several parameterized algorithms are known. We present a novel search tree algorithm for the problem, which improves running time from $O(1.76^k + m + n)$ to $O(1.62^k + m + n)$ for m edges and n vertices. In detail, we can show that we can always branch with branching vector $(2, 1)$ or better, resulting in the golden ratio as the base of the search tree size. Our algorithm uses a well-known transformation to the integer-weighted counterpart of the problem. To achieve our result, we combine three techniques: First, we show that zero-edges in the graph enforce structural features that allow us to branch more efficiently. This is achieved by keeping track of the parity of merged vertices. Second, by repeatedly branching we can isolate vertices, releasing cost. Third, we use a known characterization of graphs with few conflicts. We then show that Integer-Weighted Cluster Editing remains NP-hard for graphs that have a particularly simple structure: namely, a clique minus the edges of a triangle.

1. Introduction

Given an undirected graph G, the Cluster Editing problem asks to transform a graph by at most k edge modifications into a vertex-disjoint union of cliques. In the corresponding Integer-Weighted Cluster Editing problem, we are given integer modification costs for each edge or non-edge, and we ask if there is a set of edge modifications with total weight at most k. Here, one assumes that all edges have non-zero modification cost.

In application, the above task corresponds to clustering objects, that is, partitioning a set of objects into homogeneous and well-separated subsets. Similar objects are connected by an edge, and a cluster is a clique of the input graph. The input graph is corrupted and we have to clean (edit) the graph to reconstruct the clustering under the parsimony criterion. Clustering data still represents a key step of numerous life science problems. The weighted variant of the Cluster Editing problem has been frequently proposed for clustering biological entities such as proteins [24,25].

The Cluster Editing problem is NP-hard [19], and several proofs of this fact have been published. The Cluster Editing problem remains NP-hard on graphs with maximum degree six [18]. The parameterized complexity of Cluster Editing, using the number of edge modifications as parameter k, is well studied, see also the FPT races column in [23]. A first algorithm with running time $O^*(2.27^k)$ [12] was improved to $O^*(1.92^k)$ by an extensive case analysis [11]. By transforming the unweighted Cluster Editing problem to the integer-weighted variant, running time was advanced to $O^*(1.82^k)$ [1]. Using a characterization of graphs that do not contain many conflicts, results in the previously fastest algorithm for Cluster Editing with running time $O^*(1.76^k)$ [3]. A linear kernel with $4k$ vertices [13] was later improved to $2k$ vertices [5]. Similarly, a kernel for the integer-weighted variant with $O(k^2)$ vertices [1] was improved to $2k$ vertices [4]. Recently, Cluster Editing

* Tel.: +49 3641 946450; fax: +49 3641 946452.
E-mail address: sebastian.boecker@uni-jena.de.

1570-8667/$ – see front matter
http://dx.doi.org/10.1016/j.jda.2012.04.005

with "don't care edges" (that is, edges whose modification cost is zero) has been shown to be fixed-parameter tractable [20]. Alternative parameterizations of the Cluster Editing problem have been studied in [18]. To find exact solutions in practice, a combination of data reduction and Integer Linear Programming proved to be very efficient [2]. Related problems such as Cluster Deletion [3,7], Cluster Vertex Deletion [16], or s-plex Editing [14] have also been studied frequently in the literature.

Our contributions. We present a new search tree algorithm for Cluster Editing with running time $O(1.62^k + m + n)$ for m edges and n vertices, being the fastest known for the problem. The algorithm itself is rather simple, and is based on the merge branching introduced in [1]. The improved running time is based on three observations: All graphs created through our branching satisfy a particular property, namely, the "parity property", see Section 3. Next, as soon as we have generated zero-edges, we can continue to branch on the incident vertices until a vertex is isolated from the remaining graph, releasing cost bound by bookkeeping, see Section 4. Finally, we use a known characterization of graphs that are "locally simple", see Section 5. We stress that our result does not hold in general for the Integer-Weighted Cluster Editing problem, as such instances will usually not satisfy the "parity property". Finally, we show that Integer-Weighted Cluster Editing is NP-hard even for graphs that have a particularly simple structure, namely, a complete weighted graph minus a triangle.

2. Preliminaries

A problem with input size n and parameter k is *fixed-parameter tractable* (FPT) if it can be solved in $O(f(k) \cdot p(n))$ time where f is any computable function and p is a polynomial. In our analysis, we naturally focus on the $f(k)$ factor. For a general introduction we refer to [8,21]; in particular, we assume familiarity with bounded search trees, branching vectors, and branching numbers. In the following, let n be the number of vertices, and k the number of edge modifications.

For brevity, we write uv as shorthand for an unordered pair $\{u, v\} \in \binom{V}{2}$. Let $s : \binom{V}{2} \to \mathbb{Z}$ be a *weight function* that encodes the input graph: For $s(uv) > 0$ a pair uv is an edge of the graph and has deletion cost $s(uv)$, while for $s(uv) < 0$, the pair uv is not an edge (a *non-edge*) of the graph and has insertion cost $-s(uv)$. The *neighborhood* of u, denoted $N(u)$, is the set of all vertices $v \in V$ such that $s(uv) > 0$. If $s(uv) = 0$, we call uv a *zero-edge*. We require that there are no zero-edges in the input graph. Nonetheless, zero-edges can appear in the course of computation and require additional attention when analyzing the algorithm.

The connected components of an integer-weighted graph are the connected components of the unweighted graph that contains exactly the edges of the integer-weighted graph. We say that $C \subseteq V$ is a *clique* in an integer-weighted graph if all pairs $uv \in \binom{C}{2}$ are edges. If all vertex pairs of a connected component are either edges or zero-edges, we call it a *weak clique*. Vertices uvw form a *conflict triple* in an integer-weighted graph if uv and vw are edges but uw is either a non-edge or a zero-edge. We distinguish two types of conflict triples uvw: if uw has weight zero then the conflict triple is called *weak*, whereas if uw is a non-edge then the conflict triple is called *strong*. See Fig. 1 for examples. An integer-weighted graph is *transitive* if it is a disjoint union of weak cliques. If an integer-weighted graph contains no conflict triples then it is transitive. But the converse is obviously not true, as the example of a single weak conflict triple shows: This graph is a weak clique but contains a (weak) conflict triple. To solve Integer-Weighted Cluster Editing we first identify all connected components of the input graph and calculate the best solutions for each component separately, because an optimal solution never connects disconnected components. Furthermore, if the graph is decomposed during the course of the algorithm, then we recurse and treat each connected component individually.

An unweighted Cluster Editing instance can be encoded as an integer-weighted instance by assigning weights $s(uv) \in \{+1, -1\}$. In the resulting graph, all conflict triples are strong. During data reduction and branching, we may set pairs uv to "*forbidden*" or "*permanent*". Permanent pairs can be merged immediately: *Merging* uv means replacing the vertices u and v with a single vertex u', and, for all vertices $w \in V \setminus \{u, v\}$, replacing pairs uw, vw with a single pair $u'w$. In this context, we say that we *join* vertex pairs uw and vw. The weight of the joined pair is $s(u'w) = s(uw) + s(vw)$. In case one of the pairs is an edge while the other is a non-edge, then we can decrease parameter k by $\min\{|s(uw)|, |s(vw)|\}$. Note that we may join any combination of two edges, non-edges, or zero-edges when merging two vertices. We stress that joined pairs can be zero-edges. See again Fig. 1.

We encode a *forbidden pair* uv by setting $s(uv) = -\infty$. By definition, every forbidden pair uv is a non-edge, since $s(uv) < 0$. A forbidden pair uw can be part of a conflict triple uvw, which then is a strong conflict triple. Assume that we join pairs uv and uw where uw is forbidden and, hence, a non-edge. From the above definition, the resulting pair $u'w$ is forbidden, too, as $s(u'w) = s(uw) + s(vw) = -\infty + s(vw) = -\infty$ holds for all $s(vw) \in \mathbb{Z} \cup \{-\infty\}$. Finally, if uw is forbidden and vw is an edge then k is decreased by $\min\{\infty, |s(vw)|\} = s(vw)$. See again Fig. 1.

The following branching was proposed in [1]: We *branch on an edge* uv by recursively calling the algorithm two times, either removing uv and setting it to forbidden, or merging uv. If uv is part of at least one strong conflict triple, then merging uv will generate cost: As there is both an edge uw and a non-edge vw, we can reduce k by $\min\{s(uw), -s(vw)\}$. In case $s(uw) = -s(vw)$, joining uw and vw into $u'w$ results in $u'w$ being a zero-edge. At a later stage of the algorithm, this would prevent us from decreasing our parameter when joining the zero-edge $u'w$. To circumvent this problem, the following bookkeeping trick was introduced in [1]: We assume that joining uw and vw with $s(uw) = -s(vw)$ only reduces the parameter by $\min\{s(uw), -s(vw)\} - \frac{1}{2} = |s(uw)| - \frac{1}{2} \geqslant \frac{1}{2}$. If at a later stage we join this zero-edge with another pair,

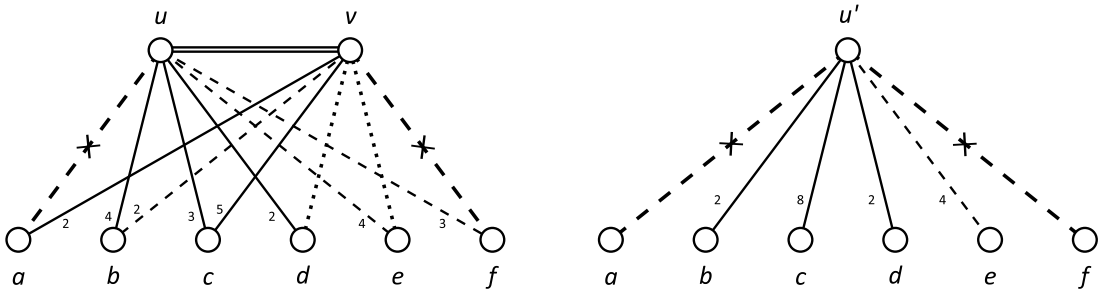

Fig. 1. Merging a pair uv. Dashed lines are non-edges, pointed lines are zero-edges. Non-edges ua, vf, $u'a$, $u'f$ are forbidden. Pairs between vertices $a, \ldots, f$ have been omitted for clarity. Conflict triples uva and vub are strong, conflict triple vud is weak. Joining ua, va decreases the parameter by 2, and so does joining ub, vb. If we use bookkeeping, then joining ud, vd and joining ue, ve both decrease the parameter by $\frac{1}{2}$ for destroying a zero-edge each.

we decrease our parameter by the remaining $\frac{1}{2}$. So, both generating and destroying a zero-edge generates cost of at least $\frac{1}{2}$. Note that joining any pair with a forbidden pair cannot create a zero-edge. See again Fig. 1.

Assume that $s(vw) = -s(uw)$ with $|s(vw)| = |s(uw)| \geqslant 2$. Then, merging an edge uv in a conflict triple uvw will also generate a zero-edge, and generates cost of at least $\frac{3}{2}$. In contrast, if $s(vw) = -s(uw) = \pm 1$ then merging uv has cost only $\frac{1}{2}$. In our analysis, we sometimes concentrate on the "hard" case that $s(vw) = -s(uw) = \pm 1$. We do so only if it is absolutely obvious that $|s(vw)| = |s(uw)| \geqslant 2$ will result in the desired branching vector.

During branching, we may reach an instance (G', k') that does not contain any (weak or strong) conflict triples. At this point, we have to "cash in" the remaining zero-edges, replacing them by non-edges. This generates cost of $\frac{1}{2}$ for each zero-edge. To this end, we check whether at most $2k'$ zero-edges are present in G'. If so, we halt the algorithm and output "yes" since G' is a solution. Otherwise, the true modification cost exceed the initial parameter k, so we discard G' and continue our search tree algorithm.

Our fixed-parameter algorithms require a cost limit k: In case a solution with cost $\leqslant k$ exists, the algorithm finds this solution and answers "yes"; otherwise, "no solution" is returned. To find an optimal solution we call the algorithm repeatedly, increasing k. Note that in truth we are interested in the solution itself; but for our formal presentation, we will usually concentrate on the algorithm answering "yes" or "no".

3. Vertex parities

We need a simple observation about the input graphs to reach an improved running time: An integer-weighted graph G with weight function $s : \binom{V}{2} \to \mathbb{Z}$ has the *parity property* if there is a *parity mapping* $p : V \to \{\text{EVEN}, \text{ODD}\}$ such that, for each pair uv, $s(uv)$ is odd if and only if both $p(u) = \text{ODD}$ and $p(v) = \text{ODD}$ holds. We ignore forbidden pairs in this definition, since $s(uv) = -\infty$ has no parity. Note that p is not necessarily unique, as demonstrated by a graph with two vertices and even edge weight. We infer a few simple observations from this definition: If $s(uv)$ is even, then either u or v or both must have EVENparity. If u is EVENthen $s(uv)$ is even or uv is forbidden, for all $v \neq u$.

Clearly, an unweighted instance of Cluster Editing has the parity property, as we can set $p(u) = \text{ODD}$ for all vertices $u \in V$. The interesting observation is that a graph does not loose the parity property if we merge two vertices. Quite possibly, this result has been stated before in a different graph-theoretical context.

Lemma 1. *Assume that an integer-weighted graph G has the parity property. If we merge two vertices in G, then the resulting graph also has the parity property.*

Proof. Assume that the weight function $s : \binom{V}{2} \to \mathbb{Z}$ encodes G, and that p is the parity mapping of G. We merge u, v and define a parity mapping p' for the resulting graph G' with $p'(w) := p(w)$ for all $w \neq u, v$, and

$$p'(u') := \begin{cases} \text{ODD} & \text{if either } p(u) = \text{ODD or } p(v) = \text{ODD} \\ \text{EVEN} & \text{if } p(u) = p(v) = \text{ODD or } p(u) = p(v) = \text{EVEN} \end{cases}$$

for the new vertex u' that results from merging u, v. It is easy to check that the resulting graph has the parity property for this mapping: For example, if $s(uw)$ and $s(vw)$ are odd then u, v, w are ODD and, in G', $s(u'w) = s(uw) + s(vw)$ is even and u' is EVEN. □

If the input graph has the parity property then, after any sequence of merging operations, the resulting graph still has the parity property. This is particularly so for the edge branching from [1], as both operations (setting an edge to forbidden, or merging two vertices) preserve the parity property. For our branching, it is important to notice that a zero-edge has even parity, so the parity of at least one of its incident vertices must be EVEN.

4. Isolation and vertices of even parity

Let $\varphi = \frac{1+\sqrt{5}}{2} = 1.61803\ldots$ be the *golden ratio*, satisfying $\varphi = 1 + \frac{1}{\varphi}$. One can easily see that a search tree algorithm with branching vector $(2, 1)$ results in a search tree of size $O(\varphi^k)$: This branching number is the positive root of $x^{-2} + x^{-1} - 1$, so $1 + x - x^2 = 0$, and dividing by x results in the definition of the golden ratio.

Our branching strategy is based on a series of lemmata, ensuring that either there is an edge to branch on, or that the remaining graph is "easy". Clearly, branching on an edge that is part of four or more conflict triples results in the desired branching number, see Observation 1 below. To this end, we concentrate on the critical case of three conflict triples. First, we consider the case of three strong conflict triples:

Lemma 2. *Let G be an integer-weighted graph that has the parity property. Assume that an edge uv is part of exactly three conflict triples, all of which are strong. Then, we can branch with branching number $\varphi = 1.61803\ldots$.*

We use the following lemma to show that we can find an edge to branch on, if we can find an edge that is part of at least three conflict triples.

Lemma 3. *Let G be an integer-weighted graph that has the parity property. Assume that an edge uv is part of three or more conflict triples. Then, we can either find an edge with branching number φ, or we can reduce k without branching.*

The remainder of this section is devoted to proving these two central lemmata. Before we do so, we collect some observations regarding certain structures in the graph that allow us to branch with branching vector $(1, 2)$ or better (that is, branching number φ or better). These observations will be used repeatedly throughout the proofs.

Observation 1. If an edge uv is part of four (weak or strong) conflict triples then it is obvious that we can branch on uv with branching vector $(1, 4 \cdot \frac{1}{2}) = (1, 2)$ or better.

Observation 2. If an edge uv with $s(uv) \geqslant 2$ is part of two or more (weak or strong) conflict triples, then branching on uv has branching vector $(2, 2 \cdot \frac{1}{2}) = (2, 1)$ or better: deleting uv costs $s(uv) \geqslant 2$ and merging uv costs at least $2 \cdot \frac{1}{2}$.

Observation 3. Assume that an edge uv with $s(uv) \geqslant 2$ is part of a (weak or strong) conflict triple uvx, and that there is an additional zero-edge uy with $y \neq x$. Then, branching on uv has branching vector $(2, 2 \cdot \frac{1}{2}) = (2, 1)$ or better: Clearly, deleting uv costs $s(uv) \geqslant 2$. But merging uv costs at least $\frac{1}{2}$ for destroying the conflict triple uvx, plus an additional $\frac{1}{2}$ for destroying the zero edge uy, releasing its cost from bookkeeping. Note that the pair resulting from joining edges uy and vy can be an edge, non-edge, or zero edge; the important point is that in all three cases, we can release the cost of zero-edge uy. Clearly, the same holds true if there is an additional zero-edge vy.

Observation 4. If an edge uv is part of three conflict triples, and there is a conflict triple uvx such that $s(vx) \geqslant 2$ or $s(ux) \leqslant -2$, then branching on uv has branching vector $(1, 2)$ or better. We stress that this includes the case where ux is a forbidden edge. Obviously, deleting uv costs at least 1. But merging uv costs at least $2 \cdot \frac{1}{2} + 1$: For $-s(ux) \neq s(vx)$ merging uv into u' will not create a new zero-edge $u'x$, so we do not have to put aside $\frac{1}{2}$. For $-s(ux) = s(vx)$ merging uv generates cost of at least $\frac{3}{2}$.

Observation 5. If a vertex v is isolated (that is, only non-edges and zero-edges are incident to v), then it is a cluster of size one in any optimal solution. In this case, we can immediately "cash" all costs of zero-edges incident to v: If there are l zero-edges incident to v then we decrease k by $\frac{l}{2}$.

We now present the proofs of Lemmas 2 and 3.

Proof of Lemma 2. We will show that we can find an edge to branch on, with branching vector $(1, 2)$ or better. In our reasoning, we will show that either, we have already reached the desired branching vector; or, we can infer certain structural properties about the instance.

Let a, b, c be the three vertices that are part of the three conflict triples with u, v. If $s(uv) \geqslant 2$ then we are done by Observation 2. If uvx with $x \in \{a, b, c\}$ is a conflict triple such that $s(vx) \geqslant 2$ or $s(ux) \leqslant -2$, then we are done by Observation 4. The same argumentation holds for a conflict triple vux. In the following, we may assume that $s(uv) = 1$ and $|s(wx)| = 1$ holds for all $w \in \{u, v\}$ and $x \in \{a, b, c\}$; for all other cases, we have already shown that the desired branching vector can be reached. This implies that u, v, a, b, c are ODD.

Assume that u, v do not have a common neighbor, so $N(u) \cup N(v) = \{u, v, a, b, c\}$. Then, merging u, v into u' generates three zero-edges $u'a$, $u'b$, $u'c$, and u' is isolated, so $N(u') = \emptyset$. By Observation 5, we do not have to use bookkeeping for these edges. So, branching on uv results in branching vector $(1, 3)$.

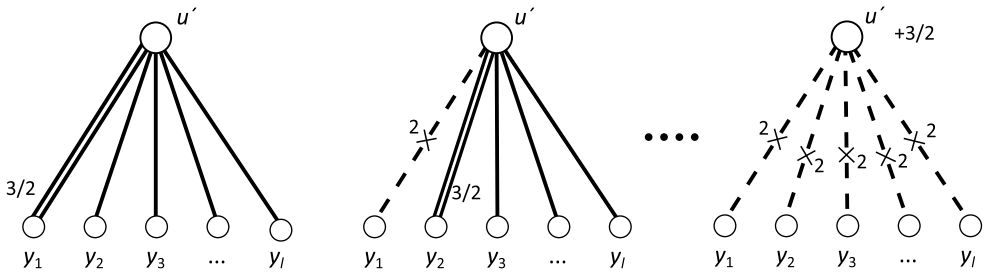

Fig. 2. Proof of Lemma 2, isolating vertex u'. Numbers indicate the cost of performing the corresponding operation. Note that merging u' with any other vertex, will destroy three zero-edges $u'a$, $u'b$, $u'c$ with cost $\frac{3}{2}$. Isolating u' also releases cost $\frac{3}{2}$ from bookkeeping.

We will now use the same trick that the merged vertex u' can be isolated, but this is slightly more involved in case u, v have at least one common neighbor. Let $Y := N(u) \cap N(v)$, then $N(u) \cup N(v) = Y \cup \{u, v, a, b, c\}$ and $|Y| \geqslant 1$. Our first step is to branch on uv: We delete uv with cost 1, and set it to forbidden.

Next, we merge u, v into a new vertex u'. This generates three zero-edges $u'a$, $u'b$, $u'c$ with cost $\frac{3}{2}$. Here, $s(u'y) \geqslant 2$ holds for all $y \in Y = \{y_1, \ldots, y_l\}$. We will now branch on all edges $u'y_j$ where the case that $u'y_j$ is deleted, is further analyzed. In detail, we either merge $u'y_i$ with cost $\frac{3}{2}$; or, we delete $u'y_i$ with cost 2 and branch on $u'y_{i+1}$, if $i < l$. See Fig. 2. Note that we either delete all edges to $y_1, \ldots, y_l$, or we finally merge some $u'y_i$ with cost $\frac{3}{2}$. In the second case, the total cost of this branch are $2(i-1) + \frac{3}{2}$. But in the very last case where all edges to $y_1, \ldots, y_l$ are deleted, we have isolated u'. Hence, by Observation 5 we can "cash" cost $\frac{3}{2}$ we have put aside when generating the three zero-edges $u'a$, $u'b$, $u'c$. So, the cost of this final branch are $2l + \frac{3}{2}$. Recall that in all cases, we have additional cost $\frac{3}{2}$ for generating the three zero-edges. In total, we reach the partial branching vector $(0+3, 2+3, \ldots, 2l+3) = (3, 5, 7, \ldots, 2l+3)$.

We combine these two partial branching vectors into one branching vector $(1, 3, 5, 7, 9, \ldots, 2l+3)$. We claim that any such branching vector corresponds to a branching number $z < \varphi$, and that the numbers converge towards φ. To this end, first note that $1/\varphi$ is the unique positive root of the polynomial $z^2 + z - 1$, that is the characteristic polynomial of branching vector $(2, 1)$. We analyze the infinite series $f(z) := z^0 + z^2 + z^4 + \cdots$ that converges for all $|z| < 1$. Now, $z^2 \cdot f(z) = f(z) - 1$ and

$$\left(z^2 + z - 1\right) \cdot f(z) = f(z) - 1 + z \cdot f(z) - f(z) = z \cdot f(z) - 1.$$

So, for the series $g(z) := z \cdot f(z) - 1$ we have

$$g(z) = z \cdot f(z) - 1 = \left(z^2 + z - 1\right) \cdot f(z)$$

and, hence, $g(1/\varphi) = 0$. For the partial sums $S_l(z) := z^{2l+3} + z^{2l+1} + \cdots + z^3 + z^1 - 1$ we infer $S_l(z) < S_{l+1}(z)$ and $S_l(z) < g(z)$ for $z \in (0, \infty)$. Also, S_l is strictly increasing in $[0, \infty)$.

Note that any polynomial of the form $p(z) := a_n z^n + \cdots + a_1 z^1 - 1$ with $a_i \geqslant 0$ for all i, has exactly one positive root for $p \not\equiv -1$. This follows as p is continuous, $p'(z) > 0$ for all $z > 0$, so p is strictly increasing in $(0, \infty)$, $p(0) = -1$, and $\lim_{z \to \infty} p(z) = \infty$. Let z_l be the unique positive root of $S_l(z)$. With $S_l(z_{l+1}) < S_{l+1}(z_{l+1}) = 0$ we finally infer

$$z_1 > z_2 > z_3 > \cdots > 1/\varphi.$$

By definition, $1/z_l$ is the branching number for branching vector $(1, 3, 5, 7, 9, \ldots, 2l+3)$, and we reach

$$1/z_1 < 1/z_2 < 1/z_3 < \cdots < \varphi.$$

Since the series S_l converges uniformly to g in the interval $[0, \alpha]$ for every $\alpha < 1$, we infer that $\lim_{l \to \infty} 1/z_l = \varphi$ must hold, which concludes the proof of the lemma. □

Proof of Lemma 3. Again, we will show that either, we have already reached the desired branching vector $(1, 2)$ or better; or, we can infer certain structural properties about the instance.

If uv is part of four conflict triples then we are done by Observation 1. If uv is part of three strong conflict triples then Lemma 2 guarantees branching number φ. So, assume that uv is part of exactly three conflict triples, and that uvw is a weak conflict triple, so uw is a zero-edge. As uv is part of exactly three conflict triples, we can choose a, b such that $N(u) \triangle N(v) = \{w, a, b\}$, where $A \triangle B = (A \cup B) - (A \cap B)$ denotes the symmetric difference of two sets A, B. If $s(uv) = 2$ then we are done by Observation 2, so we may assume $s(uv) = 1$ in the following. This implies that both u and v must have ODDparity. Since uw is a zero-edge, we infer that w has EVENparity and, hence, that $s(vw) \geqslant 2$ holds. For our worst-case considerations, we may assume $s(vw) = 2$. See Fig. 3.

If vw is part of any additional conflict triples besides wvu, then we are done by Observation 2. The same holds true if additional zero-edges besides uw are incident to v or w, see Observation 3. So, assume there are no zero-edges incident

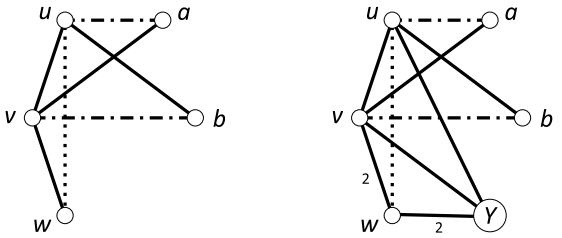

Fig. 3. Proof of Lemma 3. Dotted lines are zero-edges, dashed-dotted lines are either non- or zero-edges. Left: Initial constellation, example where va and ub are edges. Right: Including the set Y of neighbors of u, v, w.

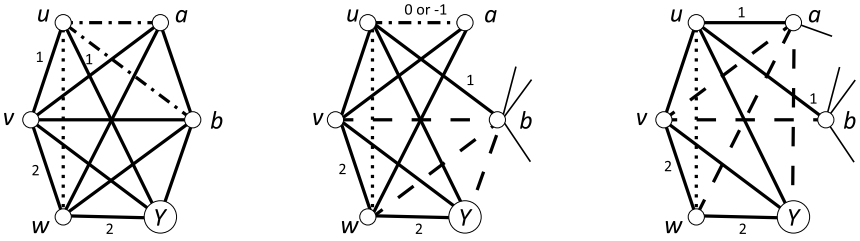

Fig. 4. Proof of Lemma 3: The three cases $a, b \in Y$ (left), $a \in Y$ and $b \notin Y$ (middle), and $a, b \notin Y$ (right). Dashed lines are non-edges, dotted lines are zero-edges, and dashed-dotted lines are either non- or zero-edges. Recall that w has EVEN parity.

to v or w besides uw, and vx is an edge if and only if wx is an edge for all $x \neq u, v, w$. Let $Y \subseteq V \setminus \{u, v, w\}$ be the set of vertices adjacent to v and, consequently, also to w. Let $X := Y \setminus \{a, b\}$, and note that this set can be empty. We infer that $X = N(u) \cap N(v)$ as otherwise, there is a fourth conflict triple for the edge uv. This implies $N(\{u, v, w\}) \subseteq \{u, v, w, a, b\} \cup X$.

Choose an arbitrary $x \in X$. If wx is part of an additional conflict triple besides wxu, then we are done by Observation 2, as w is EVENand, hence, $s(wx) \geqslant 2$. The same holds if x is incident to a zero-edge by Observation 3. Hence, we infer that each y adjacent to some $x \in X$ is also adjacent to w and, hence, $y \in Y$. So, $N(X) \subseteq \{u, v, w, a, b\} \cup X$. Furthermore, each pair $x, y \in X$ must be connected by an edge. See again Fig. 3. We distinguish three cases:

(1) Assume $a, b \in Y$, so va and vb are edges. In this case, u, v, w, a, b, X form a connected component. If ab is a non-edge or zero-edge, then we are done by Observation 3 branching on wa, since w is EVENand $s(wa) \geqslant 2$. So, we may assume that ab is an edge. By the same reasoning, ax and bx must be edges, for all $x \in X$. Next, if $s(ux) \geqslant 2$ for some $x \in X$ then we are done by Observation 3, so we may assume $s(ux) = 1$ for all $x \in X$. See Fig. 4. The cost of isolating u from all other vertices is $|X| + 1$, and the resulting graph consists of two cliques $\{u\}$ and $\{v, w, a, b\} \cup X$. The cost of any other cut in this connected component is at least $|X| + 3$ (for isolating a or b), since w is adjacent to all vertices but u with edges of weight at least 2. The cost of transforming the connected component into a clique is $|s(ua)| + |s(ub)|$. So, we can test in constant time if one of the two possible transformations has cost at most k.

(2) Assume $a \in Y$ and $b \notin Y$, so va and ub are edges. Then, $N(\{u, v, w, a\} \cup X) \subseteq \{u, v, w, a, b\} \cup X$. For $s(ua) < -1$ we are done by Observation 4 branching on uv. So, $s(ua) \in \{0, -1\}$ must hold. Since bv is a non-edge, bw and bx for all $x \in X$ are also non-edges: Otherwise we branch on vw or wx, see above and Observation 3. If $s(ub) \geqslant 2$ then we are done branching on ub, see Observation 3. See again Fig. 4. Now, one can easily check that no optimal solution can bisect v, w, a, X (that is, v, w, a, X must all be part of the same cluster in the solution): For $X = \emptyset$ a bisection of vertices v, w, a costs at least 3 since w is EVEN; and for $X \neq \emptyset$ cost are at least 4. Given a solution that bisects v, w, a, X, we modify the solution by putting u, v, w, a, X in an isolated clique, with cost at most 1 for inserting ua, and cost 1 for removing ub. Clearly, this new solution has smaller total cost than the initial solution, so the initial solution cannot be optimal. Hence, we can merge v, w, a, X without branching, generating cost of at least $\frac{1}{2}$ for destroying the zero-edge uw.

(3) Assume $a, b \notin Y$, so ua and ub are edges. Then, va and vb are non-edges, since no zero-edges can be incident to v. Similar to above, wa and wb as well as ax and bx for all $x \in X$ can be assumed to be non-edges, too: Otherwise, we can branch on vw or wx. If $s(ua) \geqslant 2$ then we are done by branching on ua, see Observation 3. So, we infer $s(ua) = 1$ and, by symmetry, $s(ub) = 1$. See again Fig. 4. Now, merging uv into some vertex u' results in a separated clique with vertex set u', w, Y that is not connected to the rest of the graph, and can be removed immediately. Hence, branching on uv leads to branching vector $(1, 2)$ as we do not have to put away $2 \cdot \frac{1}{2}$ for potentially destroying zero-edges $u'a$ and $u'b$ later.

We have shown that we can find an edge that allows for the desired branching vectors, simplify the instance and reduce k without branching, or solve the remaining instance in constant time. □

5. Solving remaining instances

Assume that there is no edge in the graph that is part of three or more (weak or strong) conflict triples. We transform our weighted graph into an unweighted counterpart G_u, where zero-edges are counted as non-existing. This graph G_u is called the *type graph* of the weighted graph. Then, there is no edge uv in the unweighted graph G_u that is part of three conflict triples, since weak or strong conflict triples in the weighted instance become conflict triples in G_u. Damaschke [7] characterizes such graphs: Let P_n, C_n, K_n be the chordless path, cycle, and clique on n vertices, respectively. Let $G + H$ denote the disjoint union of two graphs, and let $p \cdot G$ denote p disjoint copies of G. Let $G * H$ be the graph $G + H$ where, in addition, every vertex from G is adjacent to every vertex from H (graph join operation). Finally, the graph G^c has the same vertex set as G, and $\{u, v\}$ is an edge of G^c if and only if it is no edge of G. Now, Theorem 2 from [7] states:

Lemma 4. *Let G be a connected, unweighted graph such that no edge is part of three or more conflict triples. Then, G has at most six vertices, is a clique, a path, a cycle, or a graph of type $K_q * H$ for $q \geqslant 0$ and $H \in \{K_1 + K_1, C_5, P_4, K_1 + K_1 + K_1, K_2 + K_2, K_2 + K_1, (p \cdot K_2)^c\}$, $p \geqslant 2$.*

In fact, the characterization in [7] is slightly more complicated: To this end, note that $K_q * P_3 = K_{q+1} * (K_1 + K_1)$. Any non-edge in the type graph can be a non-edge or zero-edge in the weighted graph, and edges and non-edges can be arbitrarily weighted. We now show that we can efficiently solve all remaining, "simple" instances. This is similar to the argumentation in [3] but as we want to reach branching vector $(2, 1)$, our argumentation is slightly more involved.

Lemma 5. *Let G be a connected graph that has the parity property. Assume that there is no edge that is part of three conflict triples. Then, we can find an edge with branching number φ; reduce k without branching; or, we can solve the instance in polynomial time.*

Proof. If the graph is a clique, we are done. We transform our graph G into the corresponding type graph G_u, then G_u is connected and does not contain an edge that is part of three conflict triples. Lemma 4 characterizes all of these graphs. If G has up to six vertices, we can solve it in constant time. If G_u is a cycle or a path, then it can be solved in $O(n^3)$ time [1].

Next, assume that the graph is $K_q * H$ for $q \geqslant 1$ and $H \in \{K_1 + K_1, C_5, K_1 + K_1 + K_1, K_2 + K_2, P_4, K_2 + K_1\}$. We check if transforming the graph into a clique has cost at most k, in which case the answer is YES. This can be done in constant time, as there are at most five edges missing in the graph. We make a case distinction depending on H:

- Assume that $H = K_1 + K_1$, so the graph is a clique minus a single non-edge or zero-edge uv. Now, u and v are either part of the same cluster, and we insert uv with cost $-s(uv)$; or the u and v belong to different clusters, and we separate them with cost of a minimum u-v-cut. A maximum u-v-flow corresponding to the minimum u-v-cut can be computed in polynomial time [10].
- Assume that $H = K_2 + K_1$, where u, v are the vertices of the K_2 and w is the vertex of the K_1. We branch on uv: Merging uv results in a graph $K_q * (K_1 + K_1)$ we can solve in polynomial time. Afterwards, we can delete uv and decrease k by $s(uv) \geqslant 1$ without branching.
- Assume that $H \in \{C_5, K_1 + K_1 + K_1, K_2 + K_2\}$. Recall that by Observation 2, an edge uv that is part of two (weak or strong) conflict triples and has weight $s(uv) \geqslant 2$, allows us to branch with branching vector $(2, 1)$. For any vertex u of K_q and any v of H, uv is an edge and is part of two conflict triples, so we can assume $s(uv) = 1$. We have checked above if we can transform G into a clique; so, any solution partitions the vertices of H, as a partition that leaves H intact, is surely suboptimal. We claim that an optimal solution will put all vertices in K_q together with the largest cluster C from H: Putting vertices from K_q into different clusters of the solution, or putting K_q with a different cluster from H will only increase the cost. So, for each partition of vertices from H we put K_q with the largest cluster and compute the cost of the resulting partition. If we can find a partition with cost at most k then the answer is YES. As the number of partitions is constant, we can solve the instance in polynomial time.
- Assume that $H = P_4$, where u, v, w, z are the vertices of the P_4. This case is somewhat technical, and requires several steps of branching. If $s(vw) \geqslant 2$ then branching on vw has branching vector $(2, 1)$ by Observation 2, so we can assume $s(vw) = 1$, and v and w are ODD. First, we branch on vw, either deleting vw with cost 1, or merging vw into a new vertex v'. Consider the second case: If neither $v'u$ nor $v'z$ is a zero-edge, then merging vw generated cost at least 2 and we are done. Now, assume without loss of generality that $v'u$ is a zero-edge. If $v'z$ is a zero-edge, too, then uz is the only non-edge in the graph. We can compute an u-z-cut in polynomial time, and test whether the cost of this cut does not exceed the remaining parameter. The same holds true if $v'z$ is an edge, so we are left with the case that $v'z$ is a non-edge, in which case merging vw generated cost $\frac{3}{2}$. Now, v' is EVEN, so $s(v'z) \leqslant -2$ and $s(v'x) \geqslant 2$ for all vertices x of K_q must hold. We branch on $v'x$: Merging $v'x$ generates cost at least $\frac{3}{2}$ for destroying the zero-edge $v'u$ plus the conflict triple $v'xz$ with $s(v'z) \leqslant -2$. Deleting $v'x$ has cost 2, and we branch one last time: Choose another y from K_q. Merging $v'y$ generates cost $\frac{5}{2}$, deleting $v'y$ costs 2. In total, we reach the branching vector $(1, \frac{3}{2} + \frac{3}{2}, \frac{3}{2} + 2 + \frac{5}{2}, \frac{3}{2} + 2 + 2) = (1, 3, 6, 5\frac{1}{2})$ with branching number $1.604 < \varphi$.

The final case that our graph is of the type $K_q * (p \cdot K_2)^c$ with $q \geqslant 0$ and $p \geqslant 2$, is slightly more complicated. We claim that there exist no zero-edges in the graph: If there is a zero-edge vw then one of the vertices, say v, has EVENparity. But for

graphs of this type, we can always find an edge uv incident with v that is part of two conflict triples. From $s(uv) \geqslant 2$ we infer that we can branch on uv with branching vector $(2,1)$ by Observation 2. Furthermore, for any pair of non-edges, we can find an edge that is part of two conflict triples with these two non-edges. We conclude that there can be at most one non-edge vw with $s(vw) < -1$: Otherwise, choose the edge uv that "connects" the two non-edges, then branching on uv leads to branching vector $(1,2)$ as we create no zero-edges, see Observation 4. In the following, let vw be a non-edge such that all non-edges but vw have insertion cost one.

It is easy to check that if v, w end up in the same cluster of the solution, then the solution is to insert all missing edges: This follows as all non-edges have insertion cost one. To this end, inserting all edges costs $p-1-s(vw)$, and we test if k does not exceed this bound. Otherwise, assume that v, w do not end up in the same cluster of a solution. Any partition of the vertices also induces a bipartition X, Y with $v \in X$ and $w \in Y$. Without loss of generality, we assume that $|X| \leqslant |Y|$. The bipartition requires us to delete $|X| \cdot |Y|$ edges if our graph was a clique. As the graph is not a clique, there are at most $\min\{|X|, p\}$ edges that we do not have to delete. Hence, the cost of a solution that induces the bipartition X, Y are at least

$$|X| \cdot |Y| - \min\bigl\{|X|, p\bigr\} \geqslant |X|\bigl(2p+q-|X|\bigr) - |X| = |X|\bigl(2p+q-1-|X|\bigr).$$

Recall that $|V| = 2p+q \geqslant 7$ what implies $p+q \geqslant 4$. As $|X| \leqslant |Y|$ we also have $1 \leqslant |X| \leqslant \frac{2p+q}{2}$. Let $f(z) := z(2p+q-1-z)$, then $f'(z) = -2z+2p+q-1$ and $f''(z) = -2$. So, f has no minima and only one maximum. We distinguish two final cases:

- If $s(vw) = -1$, then inserting all edges costs p, and any bipartition induces larger cost $f(|X|) > p$. We only have to check the interval borders to prove this claim: For $|X| = 1$, we infer $f(1) = p+p+q-2 > p$. For $|X| = \frac{2p+q}{2}$ we have $f(|X|) \geqslant \frac{7}{2} \cdot (\frac{1}{2}(2p+q)-1) > p + \frac{7}{4}(p+q-2) > p$.
- If $s(vw) \leqslant -2$, then all edges vx must have weight one as otherwise, branching on an edge vx with $s(vw) \geqslant 2$ results in branching vector $(2,1)$. Then, isolating v costs $2p+q-2$ for edge deletions and $p-1$ for edge insertions, and the cost for isolating any other vertex, as well as the deletion cost for any bipartition with $|X| \geqslant 2$, are at least this large. To this end, for $|X| = 2$ we infer $f(|X|) = 2(2p+q-1-2) = 4p+2q-6 = 3p+q-3+p+q-3 > 3p+q-3$. For $|X| = 3$ we have $f(|X|) = 3(2p+q-1-3) = 6p+3q-9 = 3p+q-3+3(p+q-2) > 3p+q-3$. Finally, for $|X| \geqslant 4$ we infer

$$\begin{aligned} f\bigl(|X|\bigr) = |X|\bigl(2p+q-1-|X|\bigr) &\geqslant 4\left(2p+q-1-\frac{1}{2}(2p+q)\right) = 4\frac{1}{2}(2p+q-2) \\ &= 4p+2q-4 = 3p+q-3+p+q-1 > 3p+q-3. \end{aligned}$$

This concludes the proof of the lemma. □

6. A golden ratio base for search tree size

Assume that G has the parity property. We want to show that we can either find an edge to branch on with branching number φ; decrease k without branching; or, solve the remaining instance in polynomial time. If there is an edge uv that is part of at least three (weak or strong) conflict triples, we branch on this edge. By Lemma 3, doing so results in branching number φ, or we reduce k without branching, as desired. We can find an edge to branch on, in time $O(n^3)$. Similarly, we can perform all other tasks required for one step of the branching, in this time. If there is no edge uv that is part of at least three conflict triples, then Lemma 5 guarantees that we can branch with branching number φ; reduce k without branching; or, solve the instance in polynomial time. To compute minimum s-t-cuts as part of Lemma 5, we use the Goldberg–Tarjan algorithm [10] to compute a maximum s-t-flow in time $O(n^3)$, independent of edge weights. We reach:

Lemma 6. *Given an instance of the* Integer-Weighted Cluster Editing *problem with no zero-edges that satisfies the parity property, this instance can be solved in $O(\varphi^k \cdot n^3)$ time.*

Theorem 1. *Given an instance of the* Integer-Weighted Cluster Editing *problem with no zero-edges that satisfies the parity property, this instance can be solved in $O(\varphi^k + n^2)$ time.*

Proof. We can combine Lemma 6 with the weighted kernel from [4] with $O(k)$ vertices and running time $O(n^2)$. Doing so results in running time $O(\varphi^k \cdot k^3 + n^2)$. To get rid of the multiplicative polynomial factor, we want to use interleaving [22]. Unfortunately, the instances generated during the course of branching may contain zero-edges, making it impossible to directly apply the kernel from [4].

The underlying idea of the following formal construction is to "inflate" the weights of original instance to an extend that modifications with cost one are unimportant for reaching an optimal instance. This is very similar in spirit to the proof of Lemma 7 below.

To this end, assume that we have reached an instance (G,k) during the course of branching, that may possibly contain zero-edges. Let z be the number of zero-edges in G. Recall that all zero-edges in G must be resolved at some point.

This implies $z \leqslant 2k$ and, hence, for $z > 2k$ we can stop. We multiply the costs of all edges and non-edges in G by $4k+1$; and by replacing all zero-edges in G by non-edges with insertion cost one. Let $\tilde{G}$ be the resulting instance.

Consider an edge or non-edge uv of G with modification cost $|s(uv)|$ in G: The corresponding edge or non-edge uv in $\tilde{G}$ has modification cost $(4k+1)|s(uv)|$ in $\tilde{G}$. On the other hand, there are at most $2k$ zero-edges in G, so removing all corresponding non-edges in $\tilde{G}$ (each having insertion cost one) can generate at most cost $2k$ for the complete instance. To this end, any solution for G with cost k corresponds to a solution in $\tilde{G}$ with cost between $(4k+1)\cdot(k-\frac{z}{2})$ and $(4k+1)\cdot(k-\frac{z}{2})+4k$. In particular, (G,k) has a solution if and only if $(\tilde{G},\tilde{k})$ has a solution where $\tilde{k} := (4k+1)\cdot(k-\frac{z}{2})+4k$: In view of $\tilde{k} < (4k+1)\cdot(k-\frac{z}{2}+1)$ the solution for $(\tilde{G},\tilde{k})$ may not modify any additional edges or non-edges of weight larger one.

Next, we can apply the kernel from [4] to $(\tilde{G},\tilde{k})$, resulting in an instance $(\tilde{G}',\tilde{k}')$ with number of vertices linear in $\tilde{k}$. By construction, $(\tilde{G},\tilde{k})$ has a solution if and only if $(\tilde{G}',\tilde{k}')$ has a solution. Also, $\tilde{k}' \leqslant \tilde{k}$ holds.

Let $\psi(x) := \lceil (x-2k)/(4k+1) \rceil$. The kernel in [4] only deletes edges, inserts non-edges, or merges vertices. We infer that for each edge or non-edge in $\tilde{G}'$, the cost of inserting or deleting it is the sum of modification costs of edges and non-edges in $\tilde{G}$, each with positive or negative sign. Non-edges with insertion cost one can contribute at most $2k$ to the cost of an edge in $\tilde{G}'$, in either direction. Let $y := \psi(x)$, then $x \in [(4k+1)y-2k, (4k+1)y+2k]$. Hence, the modification cost x of an edge or non-edge without the contribution of non-edges with insertion cost one, equals $(4k+1)y$. We transform the instance $\tilde{G}'$ back to an instance G' by applying ψ to all edge weights. Let $z' \leqslant z$ be the number of zero-edges in G', and set $k' := \lfloor \tilde{k}'/(4k+1) \rfloor + \frac{z'}{2}$. Similar to above we infer that $(\tilde{G}',\tilde{k}')$ has a solution if and only if (G',k') has a solution.

From our above reasoning, we finally infer that (G,k) has a solution if and only if (G',k') has a solution. By construction,

$$k' \leqslant \frac{\tilde{k}'}{4k+1} + \frac{z'}{2} \leqslant \frac{\tilde{k}}{4k+1} + \frac{z}{2} < k - \frac{z}{2} + 1 + \frac{z}{2} = k+1,$$

so $k' \leqslant k$ as both are integer; and the number of vertices in G' is linear in $\tilde{k}$ and, hence, polynomial in k. □

Given an unweighted Cluster Editing instance, we first identify all critical cliques in time $O(m+n)$ for a graph with n vertices and m edges [15], and merge the vertices of each critical clique [1,13]. The resulting integer-weighted instance has $O(k)$ vertices and no zero-edges, and satisfies the parity property. Using Theorem 1 we reach:

Theorem 2. Cluster Editing *can be solved in* $O(1.62^k + m + n)$ *time.*

7. Computational complexity of graph type $K_q * (K_1 + K_1 + K_1)$

Above, we have repeatedly used that integer-weighted graphs of the type $K_q * (K_1 + K_1)$, a complete graph minus an edge, can be solved in polynomial time by computing a minimum cut. Hence, it is daunting to try the same for graph type $K_q * (K_1 + K_1 + K_1)$, a complete graph minus the edges of a triangle: Let uv, vw, uw be the three non- or zero-edges in the graph. Branch on uv by either setting it to forbidden with no cost, or inserting and merging with cost $-s(uv)$. The later results in a graph of type $K_q * (K_1 + K_1)$ that can be solved in polynomial time. For the former, we analogously branch on vw and on uw, resulting in a graph where all three non- or zero-edges have been set to forbidden. Now, solving this instance requires us to solve the integer-weighted Multiway Cut problem (also known as Multiterminal Cut problem) for three terminals, see below, which is NP-hard [6]. It turns out that these two problems can be transformed into each other with polynomial blowup. This shows that not all graph types in Lemma 4 can be solved in polynomial time.

Lemma 7. Integer-Weighted Cluster Editing *is NP-hard for graphs of the type* $K_q * (K_1 + K_1 + K_1)$, $q \geqslant 1$.

We use reduction from the unweighted Multiway Cut problem with $K = 3$ terminals:

Multiway Cut. Given an unweighted graph G and K terminals, find a minimum set of edges that separate the terminals.

Proof. Proof of Lemma 7 Given an instance $G = (V, E)$ of the unweighted Multiway Cut problem with $K = 3$ and terminals u, v, w. Set $n := |V|$. We first note that edges uv, uw, and vw are irrelevant and can be ignored: If they are present, they have to be cut in the solution. We can transform the instance G into a weighted instance $G' = (V, \binom{V}{2})$ that is a complete graph. Here, an edge e of G' with $e \notin E$ receives weight 1, whereas an edge e of G' with $e \in E$ receives weight n^2. One can easily see that this weighted instance of the Multiway Cut problem has essentially the same solution as its unweighted counterpart G: We may have to delete some of the edges with weight 1 in G', but the total weight of these edges is less than n^2. In contrast, a single additional deletion of an edge with weight n^2 must result in a suboptimal solution. Hence, the solutions will agree on all edges with weight n^2 that have to be deleted.

We now remove edges uv, uw, and vw from G', and assign insertion cost n^4 to these non-edges. Deletion costs of the remaining edges are the weights in G'. Now, G' forms an Integer-Weighted Cluster Editing instance of graph type $K_q * (K_1 + K_1 + K_1)$, $q \geqslant 1$. In this instance, a solution that inserts an edge cannot be optimal, as the cost exceed that of cutting

all edges in the instance. Hence, we have to delete edges until u, v, w are in separated cliques. But as G' is a clique minus three edges, this is equivalent to a 3-way cut of G'. We infer that an optimal solution of the Integer-Weighted Cluster Editing instance G' is also an optimal solution of the unweighted Multiway Cut instance G. As computing an optimal 3-way cut is NP-hard [6], this also holds for the Integer-Weighted Cluster Editing problem for graph type $K_q * (K_1 + K_1 + K_1)$. □

From the proof, it is obvious that Lemma 7 also holds for Integer-Weighted Cluster Deletion, where we are not allowed to insert any edges. In general, Integer-Weighted Cluster Deletion is a special case of Integer-Weighted Cluster Editing if we choose all insertion costs sufficiently large.

8. Conclusion

We have presented a parameterized algorithm for the Cluster Editing problem, that reaches the golden ratio as the base for the exponential growth of the running time. It is noticeable that search tree approaches plus additional structural observations still have a lot of potential to yield better FPT algorithms for well-known problems, even without extensive case handling. Note that the underlying edge branching is also very swift in practice, and can usually process instances with thousands of edge modifications in a matter of minutes [2].

The base $\varphi = \frac{1+\sqrt{5}}{2} = 1.61803\ldots$, resulting from branching vector $(2, 1)$, appears repeatedly in the analysis of advanced algorithms for the Cluster Editing problem [1,3]. Hence, it is an interesting question for the future if we can get beyond the $O^*(\varphi^k)$ barrier. One possible extension lies in the split-off technique introduced in [3] for Cluster Deletion, even though it cannot be directly applied, as branching on a C_4 results in branching vector $(1, 1)$ for Cluster Editing.

The even more interesting question is whether a parameterized algorithm with *subexponential* running time exists for Cluster Editing, as conjectured by Cao and Chen [4]. Unfortunately, Komusiewicz [17] and Fomin *et al.* [9] independently showed that under the exponential time hypothesis, no such algorithm can exist.

Improving upon the running time should not be problematic for the rather technical Lemma 5. Here, the open question is: Which of the special cases are polynomial-time solvable (such as $K_q * (K_1 + K_1)$) and which are not (such as $K_q * (K_1 + K_1 + K_1)$), and what FPT algorithms can be derived for the hard ones.

On the technical side, it will be interesting if vertex parities can be applied in the analysis of other (graph modification) problems. Keeping track not only of vertex parities but of the number of vertices hat have been merged during the course of branching and data reduction, may be even more powerful for an improved analysis.

Acknowledgement

I would like to thank François Nicolas for helpful discussions.

References

[1] S. Böcker, S. Briesemeister, Q.B.A. Bui, A. Truss, Going weighted: Parameterized algorithms for cluster editing, Theor. Comput. Sci. 410 (52) (2009) 5467–5480.
[2] S. Böcker, S. Briesemeister, G.W. Klau, Exact algorithms for cluster editing: Evaluation and experiments, Algorithmica 60 (2) (2011) 316–334.
[3] S. Böcker, P. Damaschke, Even faster parameterized cluster deletion and cluster editing, Inform. Process. Lett. 111 (14) (2011) 717–721.
[4] Y. Cao, J. Chen, Cluster editing: Kernelization based on edge cuts, Algorithmica (2011), http://dx.doi.org/10.1007/s00453-011-9595-1.
[5] J. Chen, J. Meng, A 2k kernel for the cluster editing problem, J. Comput. Syst. Sci. 78 (1) (2012) 211–220.
[6] E. Dahlhaus, D.S. Johnson, C.H. Papadimitriou, P.D. Seymour, M. Yannakakis, The complexity of multiterminal cuts, SIAM J. Comput. 23 (4) (1994) 864–894.
[7] P. Damaschke, Bounded-degree techniques accelerate some parameterized graph algorithms, in: Proc. of International Workshop on Parameterized and Exact Computation (IWPEC 2009), in: Lect. Notes Comput. Sci., vol. 5917, Springer, Berlin, 2009, pp. 98–109.
[8] R.G. Downey, M.R. Fellows, Parameterized Complexity, Springer, Berlin, 1999.
[9] F.V. Fomin, S. Kratsch, M. Pilipczuk, M. Pilipczuk, Y. Villanger, Subexponential fixed-parameter tractability of cluster editing, Technical report, Cornell University Library, 2011, arXiv:1112.4419.
[10] A.V. Goldberg, R.E. Tarjan, A new approach to the maximum-flow problem, J. ACM 35 (4) (1988) 921–940.
[11] J. Gramm, J. Guo, F. Hüffner, R. Niedermeier, Automated generation of search tree algorithms for hard graph modification problems, Algorithmica 39 (4) (2004) 321–347.
[12] J. Gramm, J. Guo, F. Hüffner, R. Niedermeier, Graph-modeled data clustering: Fixed-parameter algorithms for clique generation, Theory Comput. Syst. 38 (4) (2005) 373–392.
[13] J. Guo, A more effective linear kernelization for cluster editing, Theor. Comput. Sci. 410 (8–10) (2009) 718–726.
[14] J. Guo, C. Komusiewicz, R. Niedermeier, J. Uhlmann, A more relaxed model for graph-based data clustering: s-plex cluster editing, SIAM Journal on Discrete Mathematics 24 (4) (2010) 1662–1683.
[15] W.-L. Hsu, T.-H. Ma, Substitution decomposition on chordal graphs and applications, in: Proc. of International Symposium on Algorithms (ISA 1991), in: Lect. Notes Comput. Sci., vol. 557, Springer, Berlin, 1991, pp. 52–60.
[16] F. Hüffner, C. Komusiewicz, H. Moser, R. Niedermeier, Fixed-parameter algorithms for cluster vertex deletion, Theory Comput. Syst. 47 (1) (2010) 196–217.
[17] C. Komusiewicz, Parameterized algorithmics for network analysis: Clustering & querying, PhD thesis, Technischen Universität Berlin, Berlin, Germany, 2011.
[18] C. Komusiewicz, J. Uhlmann, Alternative parameterizations for cluster editing, in: Proc. of Current Trends in Theory and Practice of Computer Science (SOFSEM 2011), in: Lect. Notes Comput. Sci., vol. 6543, Springer, Berlin, 2011.
[19] M. Křivánek, J. Morávek, NP-hard problems in hierarchical-tree clustering, Acta Inform. 23 (3) (1986) 311–323.

[20] D. Marx, I. Razgon, Fixed-parameter tractability of multicut parameterized by the size of the cutset, in: Proc. of ACM Symposium on Theory of Computing (STOC 2011), ACM, 2011, pp. 469–478.
[21] R. Niedermeier, Invitation to Fixed-Parameter Algorithms, Oxford University Press, 2006.
[22] R. Niedermeier, P. Rossmanith, A general method to speed up fixed-parameter-tractable algorithms, Inform. Process. Lett. 73 (2000) 125–129.
[23] F. Rosamond (Ed.), FPT News: The Parameterized Complexity Newsletter. Since 2005, see http://fpt.wikidot.com/.
[24] T. Wittkop, D. Emig, S. Lange, S. Rahmann, M. Albrecht, J.H. Morris, S. Böcker, J. Stoye, J. Baumbach, Partitioning biological data with transitivity clustering, Nat. Methods 7 (6) (2010) 419–420.
[25] T. Wittkop, D. Emig, A. Truss, M. Albrecht, S. Böcker, J. Baumbach, Comprehensive cluster analysis with transitivity clustering, Nat. Protocols 6 (2011) 285–295.